**2048** is a single-player puzzle game created by Gabriele Cirulli[1](https://open.kattis.com/problems/2048#a0000000002). It is played on a 4×44×4 grid that contains integers ≥2≥2 that are powers of 2. The player can use a keyboard arrow key (left/up/right/down) to move all the tiles simultaneously. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move. Please observe this merging behavior carefully in all Sample Inputs and Outputs.

**Input**

The input is always a valid game state of a 2048 puzzle. The first four lines of input, that each contains four integers, describe the 16 integers in the 4×44×4grid of 2048 puzzle. The jj-th integer in the ii-th line denotes the content of the cell located at the ii-th row and the jj-th cell. For this problem, all integers in the input will be either {0, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}. Integer 0 means an empty cell.

The fifth line of input contains an integer 0, 1, 2, or 3 that denotes a left, up, right, or down move executed by the player, respectively.

**Output**

Output four lines with four integers each. Two integers in a line must be separated by a single space. This describes the new state of the 4×44×4 grid of 2048 puzzle. Again, integer 0 means an empty cell. Note that in this problem, you can ignore the part from the 2048 puzzle where it introduces a new random tile with a value of either 2 or 4 in an empty spot of the board at the start of a new turn.

|  |  |
| --- | --- |
| **Sample Input 1** | **Sample Output 1** |
| 2 0 0 2  4 16 8 2  2 64 32 4  1024 1024 64 0  0 | 4 0 0 0  4 16 8 2  2 64 32 4  2048 64 0 0 |

|  |  |
| --- | --- |
| **Sample Input 2** | **Sample Output 2** |
| 2 0 0 2  4 16 8 2  2 64 32 4  1024 1024 64 0  1 | 2 16 8 4  4 64 32 4  2 1024 64 0  1024 0 0 0 |

|  |  |
| --- | --- |
| **Sample Input 3** | **Sample Output 3** |
| 2 0 0 2  4 16 8 2  2 64 32 4  1024 1024 64 0  2 | 0 0 0 4  4 16 8 2  2 64 32 4  0 0 2048 64 |

|  |  |
| --- | --- |
| **Sample Input 4** | **Sample Output 4** |
| 2 0 0 2  4 16 8 2  2 64 32 4  1024 1024 64 0  3 | 2 0 0 0  4 16 8 0  2 64 32 4  1024 1024 64 4 |

|  |  |
| --- | --- |
| **Sample Input 5** | **Sample Output 5** |
| 2 2 4 8  4 0 4 4  16 16 16 16  32 16 16 32  0 | 4 4 8 0  8 4 0 0  32 32 0 0  32 32 32 0 |

|  |  |
| --- | --- |
| **Sample Input 6** | **Sample Output 6** |
| 2 2 4 8  4 0 4 4  16 16 16 16  32 16 16 32  2 | 0 4 4 8  0 0 4 8  0 0 32 32  0 32 32 32 |

模拟题……好不容易在比赛结束前几分钟做完了……有以下感想

模拟题一定要把模块给分好，分别最好不相关的实现。总算成功的写出一道模拟题很有成就感。

在计算单条的时候出了个问题差点翻车……就是向左划的时候是从左往右找是否有相等的，有的话把本身翻倍并把右边的变零。嗯不这么做的话就不符合样例的特征。

#include <iostream>

#include<sstream>

#include<iomanip>

#include<string>

#include<vector>

#include<stack>

#include<queue>

#include<algorithm>

#include<map>

#include<climits>

#define hash 997

#define MAX 100005

#define ll long long

using namespace std;

vector<vector<int>>origin;

vector<vector<int>>result;

void reset(vector<vector<int>>&target) {

target.resize(4);

for (int i = 0; i < 4; i++)

{

target[i].resize(4);

}

}

vector<int> process(vector<int>coming,int status)

{

if (status == 0)//左移

{

vector<int>temp;

for (int i = 0; i < coming.size(); i++)

{

if (coming[i] != 0)

temp.push\_back(coming[i]);

}

while (temp.size() != 4)

temp.push\_back(0);

for (int i = 0; i < temp.size() - 1; i++)

{

if (temp[i] != 0)

{

if (temp[i] == temp[i + 1])

{

temp[i] += temp[i+1];

temp[i+1] = 0;

//i++;

}

}

}

coming = temp;

temp.clear();

{

for (int i = 0; i < coming.size(); i++)

{

if (coming[i] != 0)

temp.push\_back(coming[i]);

}

while (temp.size() != 4)

temp.push\_back(0);

}

return temp;

}

if (status == 1)

{

vector<int>temp;

stack<int>s1;

for (int i = coming.size() - 1; i >= 0; i--)

{

if (coming[i] != 0)

{

s1.push(coming[i]);

}

}

while (s1.size() != 4)

{

s1.push(0);

}

while (!s1.empty())

{

temp.push\_back(s1.top());

s1.pop();

}

for (int i = 3; i > 0; i--)

{

if (temp[i] != 0)

{

if (temp[i] == temp[i - 1])

{

temp[i] += temp[i-1];

temp[i-1] = 0;

//i--;

}

}

}

coming = temp;

temp.clear();

{//再除一遍0

for (int i = coming.size() - 1; i >= 0; i--)

{

if (coming[i] != 0)

{

s1.push(coming[i]);

}

}

while (s1.size() != 4)

{

s1.push(0);

}

while (!s1.empty())

{

temp.push\_back(s1.top());

s1.pop();

}

}

return temp;

}

}

void output(vector<vector<int>>target)

{

for (int i = 0; i < 4; i++)

{

cout << target[i][0];

for (int j = 1; j < 4; j++)

{

cout << " " << target[i][j];

}

cout << endl;

}

}

void transpose(vector<vector<int>>&target)

{

vector<vector<int>>temp;

reset(temp);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

temp[3-j][i] =target[i][j];

}

}

target= temp;

}

void transpose\_back(vector<vector<int>>&target)

{

vector<vector<int>>temp;

reset(temp);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

temp[j][3-i] = target[i][j];

}

}

target = temp;

}

int main()

{

reset(origin);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

cin>>origin[i][j];

}

}

int status;

cin >> status;

if (status == 0)

{

for (int i = 0; i < 4; i++)

{

result.push\_back(process(origin[i], 0));

}

output(result);

}

if (status == 1)

{

transpose(origin);

for (int i = 0; i < 4; i++)

{

result.push\_back(process(origin[i], 0));

}

transpose\_back(result);

output(result);

}

if (status == 2)

{

for (int i = 0; i < 4; i++)

{

result.push\_back(process(origin[i], 1));

}

output(result);

}

if (status == 3)

{

transpose(origin);

for (int i = 0; i < 4; i++)

{

result.push\_back(process(origin[i], 1));

}

transpose\_back(result);

output(result);

}

return 0;

}